Trusted Click: Overcoming Security issues of NFV in the Cloud

Michael Coughlin
University of Colorado Boulder

Eric Keller
University of Colorado Boulder

Eric Wustrow
University of Colorado Boulder

ABSTRACT

Network Function Virtualization has received a large amount of research and recent efforts have been made to further leverage the cloud to enhance NFV. However, since there are privacy and security issues with using cloud computing, work has been done to allow for operating on encrypted data, which introduces a large amount of overhead in both computation and data, while only providing a limited set of operations, since these encryption schemes are not fully homomorphic.

We propose using trusted computing to circumvent these limitations by having hardware enforce data privacy and provide guaranteed computation. Prior work has shown that Intel's Software Guard Extensions can be used to protect the state of network functions, but there are still questions about the usability of SGX in arbitrary NFV applications and the performance of SGX in these applications. We extend prior work to show how SGX can be used in network deployments by extending the Click modular router to perform secure packet processing with SGX. We also present a performance evaluation of SGX on real hardware to show that processing inside of SGX has a negligible performance impact, compared to performing the same processing outside of SGX.

1. INTRODUCTION

Virtualization of network functions, or middleboxes, have become an increasingly important avenue of research. This field, known as Network Function Virtualization (NFV), already has a large body of work [17, 16, 8, 12, 13, 7], and calls for running middleboxes in virtual machines to leverage the flexibility and programmability of virtualization. An extension to this body of research is to move NFV applications into the cloud, as is recommended by Aplomb [18], to also leverage the manageability, performance and cost advantages of cloud computing (shown in Figure 1).

The issue with using cloud computing is that there is a risk of exposing private data, as the hosting infrastructure is not under the control of the user of the cloud resources. Research has already shown various side channels and other techniques that can be used to steal private data from cloud environments [15, 4, 21]. What is needed to address this is some way to isolate data from the physical platform so that side channels or a malicious or compromised cloud provider cannot access it.

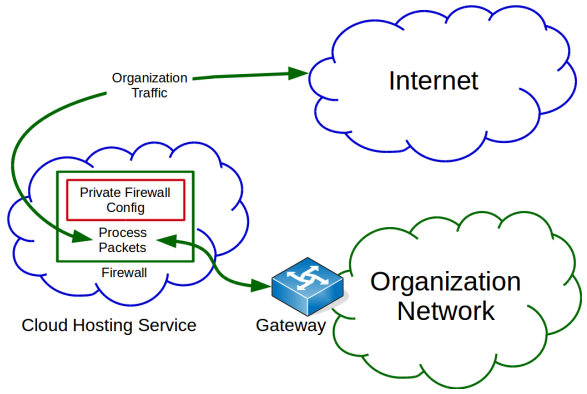

Figure 1: **Example Cloud NFV Design** — An organization exports their middlebox (*e.g.,* a Firewall) to be run in the cloud. All of the organization's traffic is tunneled to the cloud for processing before reaching the internet.

A solution to this problem would be to use fully homomorphic encryption to allow for the cloud to process this data without it ever being decrypted in the cloud. However, it is generally accepted that fully homomorphic is impractical due to its high overhead. Alternatives to full homomorphism have been proposed by prior work, which use new encryption schemes to operate over encrypted data. For example, BlindBox [19] proposes a new encryption scheme for HTTPS to allow for an Intrusion Detection System to operate over encrypted data and Embark [11] extends this further to augment virtualized network functions in general to use encryption to protect data that is processed in the cloud. However, these systems still have a large overhead and only support a limited number of operations. For example, the system proposed by Embark can only support one operation per encryption mode, but has a 4.3x data overhead for workloads that require all data to be encrypted [11].

Another alternative is to use trusted computing enforced by hardware security, such as Intel's Software Guard Extensions, as is recommended by [20]. SGX provides isolation by allowing a process to allocate an encrypted memory region that is only accessible to that process, with access enforced by the processor. This means that any data in this protected

SDN-NFV Sec'17, March 22-24 2017, Scottsdale, AZ, USA

ISBN 978-1-4503-4908-6/17/03…$15.00

DOI: http://dx.doi.org/10.1145/3040992.3040994

region cannot be accessed by any other software in the system, including other customers of the cloud service, or the service provider itself. SGX also provides a remote attestation capability, which allows for the owner of the cloud resource to verify that software is using SGX and that the correct code is protected before transmitting the secret data to this software.

Using SGX with NFV presents several challenges. The first of these is usability; SGX needs to be integrated into existing NFV development systems such that private NFV data can be secured with SGX. The second of these is performance; as SGX operates in a separate encrypted memory region, passing data to SGX and operating on this data may impose a performance penalty that needs to measured so that NFV applications can be designed to account for it. We make several contributions in this paper to address these challenges. The first of these is to describe how to modify several NFV application models to use SGX, and present an integration of SGX into Click so that arbitrary NFV applications can be supported. The second of these is a performance evaluation of SGX that compares packet processing inside and outside of SGX, to show that SGX does not impose a significant performance penalty.

The rest of this paper is organized as follows: we further motivate our architecture by examining the risks to NFV of cloud computing, and attempts by prior research in Section 2, present background information on SGX in Section 3, present our proposed architecture in Section 4, present several use cases of our architecture in Section 5, present our experimental results in Section 6, and conclude in Section 7.

2. MOTIVATION

Running virtualized network functions in a remote cloud infrastructure have been proposed by prior research, such as the system proposed by Aplomb [18]. In this system, organizations export all of their middlebox processing to a network of virtualized network functions run in cloud hosting service (*e.g.,* Amazon EC2), and tunnel traffic between the cloud and their network using a gateway. This system has many benefits for management and reliability, but brings increased risk of exposure of sensitive data. This is because using an external cloud increases exposure risk, as the infrastructure is maintained by a cloud service provider, and possibly a separate network function service provider, that can inspect or expose data. There is also a risk of data being accessed from co-located tenants (other clients of the cloud hosting service) or from security vulnerabilities in the hosting platform, as neither of these risks are in the control of the owner of the virtualized network function.

2.1 Example Private Data at Risk

These risks pose a threat to organizations that outsource their network functions, as these functions contain sensitive data about these organization's networks and business interests. For example, these common network functions include sensitive data about organizations:

- **Firewalls**: These devices protect an organization's network from external access. Sensitive information includes allowed and blocked traffic and possible information about the network topology.
- **Intrusion Detection Systems**: IDS detection rules are sensitive, as they reveal how traffic is being monitored, including what patterns are being searched for. Such information can be useful to malicious parties attempting to gain access to the network.
- **Routers**: Routing information is very sensitive, as it reveals details of the business relationship of organizations with their service providers. If revealed, this could lead competitors to exploit this information or to increases the operating costs of the organizations if there are exploitable agreements.
- **VPN Gateways**: VPNs need to have the neccessary cryptographic keys in order to create encrypted tunnels into the orgainization's network. Therefore, exposing these keys could allow for decryption of VPN traffic or even worse attacks on VPNs.

2.2 Prior Research

Prior research has already acknowledged the risks of using NFV in cloud environments. For example the authors of Aplomb noted the security risks to organizations that opt to perform their middlebox processing in the cloud [18]. The authors also note that these risks have not slowed the use of cloud computing, and using NFV in the cloud will only appear as a risk to organizations that do not trust cloud computing in general. However, there has a large body of research into information leakage via side channels and vulnerabilities in the cloud, as the economy of scale used in cloud computing generally necessitates the sharing of resources between different customers (tenants) [15, 4, 21]. As mentioned above, malicious tenants can make use of these side channels to access private data from other tenants, such as sensitive data about an organization's network or traffic. In addition, there are still the risks of interacting with a malicious cloud provider.

As a response to these privacy risks, work has been done to augment cloud applications [14] and middleboxes [19, 11] to operate on encrypted data to reduce the amount of sensitive data that the hosting cloud service has access to. By operating on encrypted data, a virtualized middlebox run in a cloud service would not have access to the plaintext data that it is processing, meaning that any malicious tenant or service provider would not be able to steal any of this data. Both BlindBox [19] and Embark [11] propose encryption schemes to increase the privacy of middleboxes that operate on untrusted infrastructure (such as a cloud service). The main drawback to operating over encrypted data is that there is still a large overhead (though less than fully homomorphic encryption), and only a limited set of operations can be performed on the data (because the encryption is not fully homomorphic).

2.3 Trusted Computing for the Cloud

To protect data in a cloud environment, there must be some capability to operate on data without exposing it to an untrusted cloud, including the operating system and physical infrastructure (*i.e.,* software isolation), and an ability to send this data without it being exposed (*i.e.,* trusted communication). Homomorphic encryption and operating on encrypted data provide these capabilities, but do so with high overhead and limited functionality (in the case of operating on encrypted data). An alternative would be to use trusted computing to provide these capabilities. For example, Trusted Platform Modules (TPMs) provide limited software isolation using remote attestation to prove that the

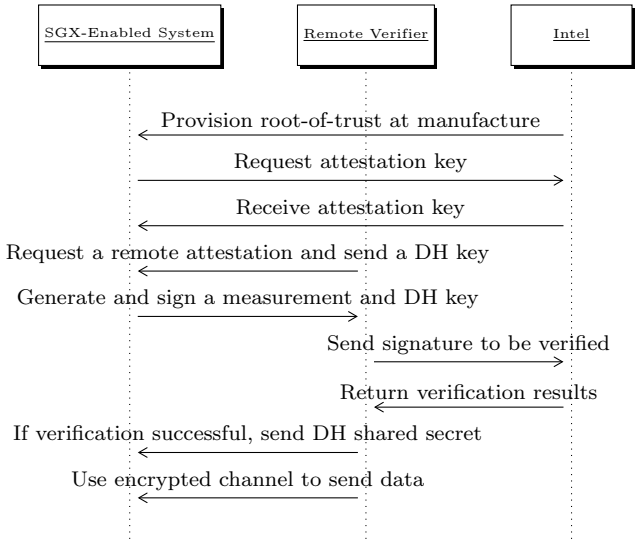

Figure 2: **Intel Remote Attestation**

software and operating system are in a secure state (the limitation is that TPMs cannot protect against access from hardware or side channels), and provide secure communication by allowing for secure storage of keys that can be used to establish encrypted communication channels [9, 10]. Intel's recent Software Guard Extensions (SGX) provide similar functionality to TPMs, but with increased protections, as the software isolation mechanism provided by SGX protects against untrusted hardware, as well as untrusted software, and the remote attestation system can be used to establish secure communication.

SGX is further explained in Section 3, but in summary SGX allows for the creation of an encrypted memory region (an enclave) to be created to execute trusted code, and for remote attestation of the code in this enclave to be performed. SGX allows for a virtualized middlebox to be designed that processes all sensitive data inside of an enclave, and to prove to the owner of the middlebox that the enclave was created correctly before transmitting data to into it, as is done by [20]. This data can be encrypted to the enclave so that only the owner and the enclave has access to it. Such a system would avoid the disadvantages of operating on encrypted data while still achieving the privacy properties of concealing the sensitive data from the cloud service and other tenants. The challenges to using SGX are designing applications such that all sensitive data is protected by SGX and integrating SGX into applications without compromising performance. We overcome the first challenge in Section 4 by presenting how to design applications to use SGX, including an integration with Click, and overcome the second challenge in Section 6 by showing that SGX does not impact performance in NFV applications.

3. SGX BACKGROUND

Intel's Software Guard Extensions (SGX) provide software isolation and remote software attestation capabilities as CPU instructions, with the root-of-trust for this attestation built into the CPU. The software isolation capabilities allow for software to have private data in memory that cannot be accessed by another process, even in the face of a root-level exploit, as access is enforced by the processor. The remote attestation feature allows for a remote verifier to confirm that software is executing in one of these protected memory regions. When combined together, a remote party can verify that certain software is running in protected memory before transmitting private data to it, and can then be sure that once received, the private data cannot be accessed by the rest of the remote system (even the operating system).

3.1 SGX Software Isolation

SGX provides new instructions that allow for a process to request an "enclave", which is a small region of memory that is only accessible to that process. The processor enforces access to this memory by encrypting all memory contents before they leave the CPU and only allowing the calling process to call code in this memory region.

By creating this enclave, a process can protect a certain amount of code and data from being accessed by other processes. Since this access is enforced by the processor, even a root-level exploit of a malicious operating system cannot access the protected region to access data or interfere with the code execution. This comes with several caveats however. First, applications need to request the operating system to allocate an enclave, which allows for a malicious operating system to perform denial-of-service on access to enclaves. Second, the code that runs inside of the enclave cannot be encrypted, so any private data needs to be sent to the enclave from outside of the system for it to be secure. Fortunately, the SGX remote attestation systems allows for a remote verifier to determine if an enclave was created and to simultaneously establish a shared secret. Finally, since SGX also requires operating system participation, applications also must rely on the Intel SGX SDK, platform software and driver to create enclaves and to implement the remote attestation system (as described below, the remote attestation flow relies on a special system enclave provided by Intel). The SGX SDK has a number of limitations as well, including the limitation that only static libraries can be linked into code that runs inside of the enclave and that the maximum memory usage of the enclave must be determined when the enclave is compiled.

3.2 SGX Remote Attestation

Remote attestation allows for a remote verifier to determine if an enclave was established on a particular SGX-enabled system and to know what code is running in this enclave. This is achieved using a challenge-response protocol to produce a measurement of the enclave that is signed by the processor, which can be verified by interacting directly with Intel. This relies on a trust chain rooted in the processor (and processor microcode, which must be signed by Intel in order to be updated [5]). A diagram of the remote attestation flow is shown in Figure 2.

In this protocol, Intel first needs to provision processors with the needed root-of-trust. Intel provisions several root keys at manufacture to bootstrap the root-of-trust, and these keys are later registered with the Intel services by users to retrieve the needed keys to perform attestations. Using these new attestation keys, trusted SGX enclaves provided by Intel can generate and sign measurements, and these signed measurements can be transmitted to a remote party when a remote attestation is requested. This remote party then verifies the measurement and has Intel verify the signature.

If both checks pass then the remote party can trust the enclave. During this process, the remote verifier and the enclave can exchange Diffie-Hellman keys to establish a shared secret ans use it to create a secure communication channel.

It should be noted that both the software isolation system and the remote attestation system only provide protections against the cloud service provider (*e.g.,* Amazon EC2 and any co-located tenants), but not against Intel, as Intel must participate in the remote attestation process. Therefore, the trust model of SGX always trusts Intel, but does not trust the operator of the physical SGC infrastructure (*i.e.,* the cloud service provider).

4. TRUSTED CLICK ARCHITECTURE

To demonstrate that SGX is practical with NFV applications, we extended the Click modular router [13] to perform packet processing in an SGX enclave (*i.e.,* a "trusted" Click processing module). Click, in summary, is a tool to allow for composing common network processing primitive elements to create larger functions, such as the creation of a IPv4 router from network interfaces, queuing elements, IPv4 classifiers and other elements. In principle, any Click element that receives a packet can export processing to SGX using a single function call, meaning that the modification or creation of Click elements is relatively simple. However, to support this, a custom external library needs to be created that implements the SGX code and Click must link to this library. Inside of this library, code must exist to load the enclave and to pass data to an enclave (in our implementation, the pointer to packet data and the packet length is passed to the enclave). Finally, the enclave must accept this data and perform the processing. We present a performance comparison of using Click with SGX in Section 6. When designing cloud middleboxes, we assume the use of the Aplomb model, that includes a gateway to tunnel traffic to the cloud middleboxes.

When using Click, packet data is passed through a graph of Click elements (collectively, this graph implements a middlebox), starting from the packet source (*e.g.,* a packet generator or network interface). However, since we are using SGX to protect the packet data, all the packets will be encrypted, meaning that each Click module protected by SGX will need to decrypt the packets for processing and reencrypt before passing to the next element. To do so, the Aplomb gateway will first need to transmit a decryption key to the SGX-protected elements, which needs to be done over a secure channel.

To establish a secure channel, the gateway will perform a remote attestation of the Click element, using the protocol shown in Figure 2, with the gateway taking the role of the remote verifier and the Click element taking the role of the SGX-enabled system (communication will be performed using an insecure channel, such as the transport provided by Click to run packets through the Click graph).

The first steps for the gateway are to generate a Diffie-Hellman (DH) key pair and transmit this key with a request for a remote verification to the middlebox. The untrusted portion of the midlebox will receive this and call an enclave ecall function (an ecall is SGX SDK terminology for a function that is executed inside of an enclave that can be called from outside of the enclave). This function will generate a DH key pair for the enclave and request a measurement from the trusted Intel Quoting enclave, and then transmit

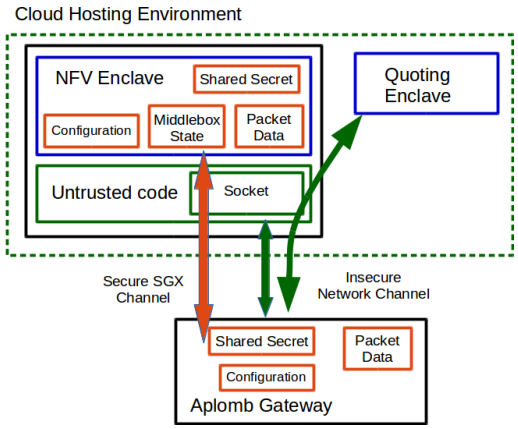

Figure 3: **NFV Swith SGX Data Ownership** — An NFV application augmented with SGX running in a cloud environment. Green segments are untrusted portions, including code not in the enclave and the hosting environment itself. Red segments are private data to the owner of the system, and the black segments are the actual applications deployed by the owner. Finally, the blue segments represent the SGX protection boundary. Note that communication of untrusted data occurs over an untrusted network channel. Using the shared secret, the gateway communicates private data with the enclave over an overlayed secure channel.

the signed quote and DH key back to the gateway using an call to an untrusted ocall (an ocall is SGX SDK terminology for a function that executed outside of an enclave, but is called from inside of an enclave), which will send the data back over an untrusted socket.

The gateway will receive the enclave's key and the signed measurement and will first verify the measurement against a pre-computed value (this value will be generated by the SGX SDK tools when an enclave is first compiled). Then, to verify the signature, the gateway will transmit the signature to Intel for verification. If the measurement matches the expected value, then the enclave in the cloud is running the expected code, and if Intel verifies the signature successfully, then the measurement was generated by a trusted SGX system, and so can be trusted as well. At this point, the enclave and the gateway can compute the shared DH key and overlay an encrypted channel over the untrusted channel by encrypting traffic (such as packet data or configuration) with this key (the untrusted code can only refuse to provide transport for this entire process, not compromise the security of the communication). Using this secure channel, the gateway can send a decryption key to the enclave in the Click module, which can use this key to decrypt the encrypted packet data that needs to be processed.

5. CLOUD NFV USE CASES

Using SGX in applications have been well studied [2, 20, 1], and at a high-level, using SGX entails partitioning applications into secure and non-secure portions, as is done by S-NFV [20] and as we describe in context of Click in Section 4. The secure portion of the application is run inside of an SGX enclave and is safe to store and operate on private data, whereas the non-secure portion has no protections from SGX. In this section, we go beyond describing how to design programs to use SGX to describing how NFV applications that use SGX should be incorporated into networks. To do this, we examine three potential use cases that

can benefit from using SGX: operating NFV applications in a cloud environment, operating on edge devices (*e.g.,* mobile phones or customer premises equipment) and operating external services in internal networks.

5.1 NFV in the Cloud

For organizations that deploy NFV applications in the cloud, we assume the model proposed by Aplomb [18]. In this architecture, an Aplomb gateway tunnels traffic to an organization's network of NFV applications in the cloud, and some (or all) of these applications use SGX to protect and store private organization data, as in [20]. However, this data is extended beyond middlebox state to include packet data and middlebox configuration, as private information about the organization can exist in this data. To secure this data, the Aplomb gateway would be extended to transmit it to the SGX enclaves in the NFV cloud network using an encrypted channel, which is established using a remote attestation (the gateway is the remote verifier, and the middlebox is the SGX-enabled system). Once data is transmitted through this channel, it can be processed securely by the SGX enclaves, and is never available in a decrypted form outside of an enclave (the Aplomb gateway can also provide the needed configurations for enclaves to establish secure channels between themselves). An illustration of where secure and insecure components in this design are located is available in Figure 3.

5.2 NFV in Edge Computing

NFV in edge computing can also benefit from SGX, as it allows for services to verify that some computation is occurring, especially given the recent trend towards edge computing. By being able to perform trusted computing, useful network services such as NAT and QoS [6], or even trusted caching of content for various streaming services can be provided without the application provider exposing sensitive data about the application. SGX has also been leveraged to achieve this [3], and can be incorporated into a larger NFV network by establishing encrypted channels between the SGX enclaves in the edge applications with the organization's internal or cloud hosted NFV network. The channel establishment is performed using the same steps as the in the cloud case, but the application takes on the role of the remote verifier and the edge device is the SGX-enabled system. The application needs to verify that the correct code is running on end devices before transmitting application configuration data to these devices, with is done using this remote attestation process.

5.3 NFV in Deployed Services

Finally, the usage model of SGX can be reversed to allow for external network function services to be provided by a service provider and hosted inside of an organization's network (*i.e.,* on premise). In this case, the organization would provide an SGX platform to allow for the external service to execute their software on. However, a remote attestation can be performed by both the external service and the hosting organization to prove that some computation is occurring. This allows for the external service provider to be sure that their product is deployed correctly and for the hosting organization to sure that billing is occurring correctly. This is because the software being attested can be proven to both parties, so any billing for provided service can be proved, as the service is attested to both parties. In this case, the SGX-enabled system is provided by the organization, and both the organization and the third party take on the role of the verifier. However, in the organization's case, the secure channel will not be used to send any data, as the enclave will only accept data received from the third party.

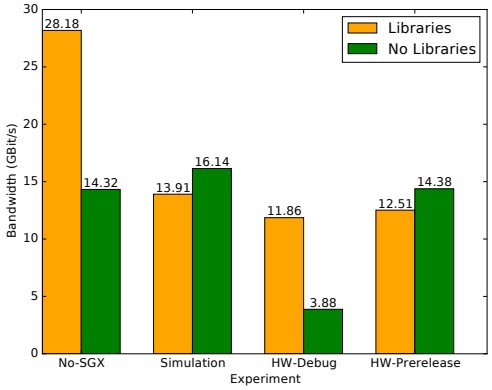

Figure 4: **Pattern Match Experiment**

6. EXPERIMENTAL RESULTS

We performed a simple experiment to determine the overhead of using SGX in a virtualized network function. As described in Section 4, we extended Click to perform packet processing in an SGX enclave. Specifically, we created custom Click elements that implemented a simple pattern search of packets to test the memory access and CPU performance of the enclave. These elements processed packets generated using Click's built-in packet generator (the InfiniteSource element), and were benchmarked against each other in various SGX compilation configurations, as well as against a baseline experiment where no processing occurred. Multiple experiments are needed, as an SGX enclave can be compiled in multiple release modes, as well as using a simulation (the SGX SDK currently only allows for enclave to be compiled in debug or prelease modes currently).

We ran two versions of this experiment using an implementation that uses provided libraries, and one that uses un-optimized code without libraries, as different libraries need to be used inside versus outside of the enclave, as described in Section 3. This is reflected in Figure 4, with the performance of the library code marked in orange, and the non-optimized code marked in green. The non-SGX implementation achieves a higher performance in the case of the orange bars due to the limitations of the SGX SDK, as code in the enclave must use libraries provided by the SDK, which are not optimized due to its immaturity. The green bars show that when executing the same non-optimized code without library dependencies, SGX itself does not impose a performance penalty, and we expect that the performance of the SGX SDK will improve as SGX matures.

We used the Intel NUC6i7KYK system, which contains an 2.5 GHz i7-6770HQ Skylake processor (which includes SGX instructions) and 16 GB of RAM.

Our experiment's implementation searches each packet passed through the element for a simple byte pattern. In this experiment, we generated 40 KB packets using the Click InfiniteSource element and passed these through the process-

ing element, in four different configurations- processing without SGX, SGX in simulation mode, SGX in hardware debug mode and SGX in hardware prerelease mode. These results are summarized in Figure 4, with the average throughput achieved outside of SGX being 14.77 GBit/s without libraries and 28.18 Gbit/s with libraries, and the best performance in SGX hardware being 14.57 GBit/s in SGX prerelease mode with libraries and 12.51 Gbit/s with libraries. From this data, SGX appears to only impose a negligible overhead for throughput when run without libraries, but only 45% efficient with libraries. This appears to be caused by optimizations that exist in the system libraries that are not present in the SGX-provided libraries (such as compiler or assembly optimizations).

7. CONCLUSION AND FUTURE WORK

In this paper, we present a potential architecture for using SGX to increase privacy of NFV applications. Compared to solutions that operate over encrypted data, SGX provides the same privacy guarantees without the data overhead and limited operations. Also, we present a benchmark of SGX being used in example middlebox processing to show that the computational overhead of using SGX versus traditional execution is negligible in a realistic middlebox development scenario. So long as the middlebox developer is aware that data is only secure when it is stored inside of an SGX enclave and makes use of remote attestation to establish a secure channel between the enclave and the rest of the network (*i.e.,* the Aplomb gateway), any private data that the middlebox needs to access will be protected. This means that SGX is a valid alternative to other systems for protecting private data in NFV applications. For future work, we intend to implement the extensions to the Aplomb gateway to interact with SGX-protected Click elements, investigate other NFV applications that could benefit from SGX, and experiment with high-speed I/O in Click, such as using DPDK.

Acknowledgments: This work was supported in part by NSF SaTC grant number 1406192.

8. REFERENCES

[1] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. OâĂŹKeeffe, M. L. Stillwell, et al. Scone: Secure linux containers with intel sgx. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Savannah, GA*, 2016.

[2] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. *ACM Trans. Comput. Syst.*, 33(3):8:1–8:26, Aug. 2015.

[3] K. Bhardwaj, M.-W. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan. Fast, scalable and secure onloading of edge functions using airbox. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 14–27. IEEE, 2016.

[4] S. Bugiel, S. Nürnberger, T. Pöppelmann, A.-R. Sadeghi, and T. Schneider. Amazonia: When elasticity snaps back. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 389–400, New York, NY, USA, 2011. ACM.

[5] V. Costan and S. Devadas. Intel sgx explained. Technical report, Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint. iacr. org/2016/086.

[6] C. Dixon, A. Krishnamurthy, and T. E. Anderson. An end to the middle. In *HotOS*, volume 9, pages 2–2, 2009.

[7] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 533–546, Berkeley, CA, USA, 2014. USENIX Association.

[8] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. Opennf: Enabling innovation in network function control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 163–174, New York, NY, USA, 2014. ACM.

[9] T. C. Group. Trusted Platform Module Main Specification (TPM1.0). http://www.trustedcomputinggroup.org/resources/tpm_main_specification, March 2011.

[10] T. C. Group. Trusted Platform Module Library Specification (TPM2.0). http://www.trustedcomputinggroup.org/resources/tpm_library_specification, March 2013.

[11] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu. Embark: securely outsourcing middleboxes to the cloud. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 255–273, 2016.

[12] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. Clickos and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 459–473, Berkeley, CA, USA, 2014. USENIX Association.

[13] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18:263–297, 2000.

[14] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.

[15] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[16] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[17] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi. The middlebox manifesto: Enabling innovation in middlebox deployment. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 21:1–21:6, New York, NY, USA, 2011. ACM.

[18] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[19] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 213–226. ACM, 2015.

[20] M.-W. Shih, M. Kumar, T. Kim, and A. Gavrilovska. S-nfv: Securing nfv states by using sgx. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 45–48. ACM, 2016.

[21] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 313–328, Washington, DC, USA, 2011. IEEE Computer Society.